Master's Thesis

QuickTalk: An Association-Free Communication Method for IoT Devices in Proximity

Seongmin Ham

Department of Computer Science and Engineering

Graduate School of UNIST

2017

QuickTalk: An Association-Free Communication Method for IoT Devices in Proximity

Seongmin Ham

Department of Computer Science and Engineering

Graduate School of UNIST

QuickTalk: An Association-Free Communication Method for IoT Devices in Proximity

A dissertation
submitted to the Graduate School of UNIST
in partial fulfillment of the
requirements for the degree of
Master of Engineering

Seongmin Ham

1. 11. 2017

Approved by

Advisor
Kyunghan Lee

QuickTalk: An Association-Free Communication Method for IoT Devices in Proximity

Seongmin Ham

This certifies that the dissertation of Seongmin Ham is approved.

1. 11. 2017

signature

Advisor: Kyunghan Lee

signature

typed name: Changhee Joo

signature

typed name: Hyoil Kim

Abstract

IoT devices are in general considered to be straightforward to use. However, we find that there are a number of situations where the usability becomes poor. The situations include but not limited to the followings: 1) when initializing an IoT device, 2) when trying to control an IoT device which is initialized and registered by another person, and 3) when trying to control an IoT device out of many of the same type. We tackle these situations by proposing a new association-free communication method, QuickTalk. QuickTalk lets a user device such as a smartphone pinpoint and activate an IoT device with the help of an IR transmitter and communicate with the pinpointed IoT device through the broadcast channel of WiFi. By the nature of its association-free communication, QuickTalk allows a user device to immediately give a command to a specific IoT device in proximity even when the IoT device is uninitialized, unregistered to the control interface of the user, or registered but being physically confused with others. Our experiments of QuickTalk implemented on Raspberry Pi 2 devices show that the end-to-end delay of QuickTalk is upper bounded by 2.5 seconds and its median is only about 0.74 seconds. We further confirm that even when an IoT device has ongoing data sessions, QuickTalk can still establish a reliable communication channel to the IoT device with little impact to the ongoing sessions.

Contents

List of Figures

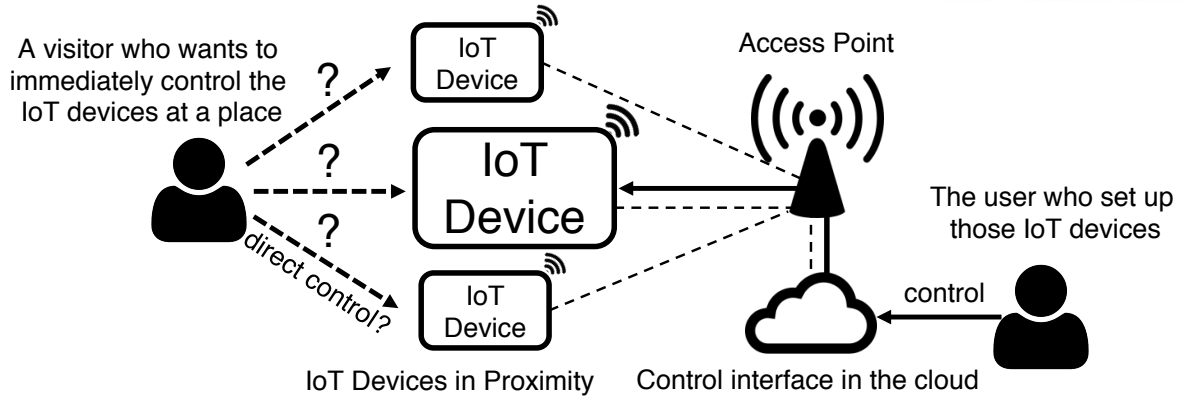


Figure 1: Controlling the IoT devices in proximity is not always straightforward.

1 INTRODUCTION

In the last few years, IoT (Internet of Things) has been arguably one of the most commercially promoted technical terms in the field of computer networks. Comparing to its early stage where IoT was just a concept of connecting numerous small devices such as sensors, actuators, and embedded systems to Internet, now it has become much more mature with a number of in-situ realizations. Such realizations that are often found in the area of home automation include thermal controller [8], wattage monitor [9], gas valve [7], and lighting controller [10]. These IoT devices are distinguished from their conventional forms by having not only the ubiquitous accessibility but also the software control interface that guarantees virtually the same or even improved usability compared to local transactions. Thanks to these properties, IoT devices are considered to be convenient and easy to use.

However, we find that there are critical situations where the recognition of the high usability of IoT devices becomes untrue. The critical situations and the problems therein are revealed by the following use cases: a) Alice visited her parents' house and found an IoT thermal controller. She wanted to change the setting of the device but noticed that she cannot do anything without having the smartphone of her mother, which previously set up the device, b) Bob who manages a restaurant bought a bunch of IoT bulbs that can dynamically change brightness and color by programmed presets, but he realized that controlling the bulbs individually is painful since putting separate names on all the densely installed bulbs in a control interface and memorizing the names is far from being intuitive, and c) Charlie who wants to publicize information such as air quality and traffic situation through outdoor IoT sensors planned to distribute these information to the passing-by users. But he realized that there is no way to directly deliver information to the users and the only solution is to display a web address or a QR code [11] to access at each sensor, which is never friendly to the passing-by users. These problematic situations are abstracted in Figure 1.

Situation a) points out that most IoT devices are only controllable by the user device which was used to set up the IoT devices. As this situation exemplifies, even though she is one of the persons who are authorized to control that IoT device, her smartphone is useless as a controller of that IoT device. It is counter-intuitive to most non-tech savvy users.

Situation b) brings up a naming challenge. When there are only few IoT devices of the same type in a place, naming is not an issue. However, as shown in the scenario, if a user has to control a set of bulbs, for instance a hundred, that are installed closely to each other, a typical naming scheme such as *bulb:1* or *bulb:living-room* no longer works. Given the widely agreed future of IoT environments that are of high-density deployments, the difficulty in controlling the devices by their names would be more prominent in the near future.

Situation c) pulls out a more technical issue in which an IoT device that is already equipped with a communication chipset such as WiFi is incapable of directly communicating with a user. This happens because the WiFi of an IoT device is occupied by the purpose of communicating with its control interface and is not listening to the users in proximity. There is no practical solution to this matter.

From the aforementioned problem statements, we observe that these problems root from a single cause, that is lack of a feature which allows a user device to communicate with a specific IoT device without going through an association process. We call this feature *association-free communication* for the IoT devices in proximity. Once this feature is enabled, a user no longer suffers from the exemplified situations. However implementing the feature brings new technical challenges: 1) how to pinpoint a device in proximity and 2) how to set up a communication channel without an association process while preserving existing sessions if there is any. We tackle these challenges by proposing *QuickTalk* that uniquely combines IR (Infrared) signal emission and WiFi overhearing over the broadcast channel.

In a nutshell, QuickTalk at a user device utilizes IR to pinpoint and trigger an IoT device and to deliver the ID of the user device (e.g., WiFi MAC address). Upon reception of the ID, the IoT device keeps broadcasting the ID through its WiFi interface to its current WiFi channel so that the user device can detect the channel of the IoT device by extracting the ID while sweeping the WiFi channels. Once the channel is known, QuickTalk lets them to communicate with each other by WiFi broadcasts at that channel.

We implement QuickTalk as software stacks for Raspberry Pi 2 devices that emulate a user device and an IoT device with IR and WiFi interfaces. Our validation reveals that thanks to the strong directivity nature of IR, QuickTalk can pinpoint and trigger an IoT device almost immediately with a narrow angle of ± 10 degree and also thanks to the nature of broadcast communication, QuickTalk allows ongoing communications at the IoT device, if any, to coexist with the newly established broadcast communication. These advantages make QuickTalk to be applicable to the IoT devices that are fresh out of the box and

even to the IoT devices that are densely deployed.

2 RELATED WORK

There exist a huge number of studies in the context of designing IoT systems. In this section, instead of providing a broad introduction, we give our focus to the previous studies that are highly relevant to our work in the following two aspects: 1) networking architecture and 2) control interface for IoT devices.

2.1 Networking Architecture

Early-stage IoT devices that showed little difference to the nodes of sensor networks were mostly relying on synchronization-based networking methods, where each device accumulates its sensor readings and periodically synchronizes the readings to a local or a remote server in batches [14]. Guinard et al. [19] pointed out the networking inefficiency (e.g., overhead, data freshness) of using synchronized-based methods in IoT systems and proposed a resource-oriented networking architecture which conforms to the principles of REST (Representational State Transfer) and utilizes embedded HTTP (Hypertext Transfer Protocol).

Later, IETF 6LoWPAN working group [16] raised performance issues of using HTTP or other TCP-based protocols in the typical environment of operating IoT systems, which is highly constrained mainly due to instability of network links, limited computing capability, and relative small battery capacity of IoT devices.

To alleviate those issues, a light-weight protocol, CoAP (Constrained Application Protocol) was proposed [25] and then was standardized by IETF (Internet Engineering Task Force) as RFC 7252 [24]. An experimental study by Leva et al. [22] showed that CoAP is indeed light-weight by demonstrating that an IoT device can save about 70% of maintenance cost of battery when using CoAP compared to using HTTP with total cost of ownership model. To achieve its goal, CoAP is designed to use UDP (User Datagram Protocol), to be RESTful, and to be easily translatable to HTTP. Although CoAP allows direct communication between IoT devices and their user devices, RFC 7252 suggests to use CoAP mainly between the IoT devices and a HTTP proxy that serves as a gateway or a control hub for the user devices. Because a HTTP proxy and user devices can communicate with HTTP in a regular manner, when a HTTP proxy exists, it is not essential for the user devices to understand CoAP. A HTTP proxy can be typically placed in the same network where IoT devices belong to but it is possible for the proxy to be placed in a cloud platform to enable ubiquitous accessibility toward IoT devices. Placing a HTTP proxy in a cloud is proposed by Kovatsch et al. [21] and this idea is now implemented in various commercial cloud platforms such as Microsoft Azure [15], Amazon Web Service [5], and Apple iCloud [12].

2.2 Control Interface

Recent commercial IoT devices are mostly designed to be controlled by a web (e.g., Google The Physical Web) or an application (e.g., Apple iHome) interface. Such interfaces are provided by the computational capability of a HTTP proxy (or a control hub) which is located either at a local server (e.g., inside a WiFi access point) or at a cloud platform. To hook up a new IoT device to such an interface, a user in general is required to operate a simplistic web interface pre-installed in an IoT device through an open WiFi connection [1]. Once the user interface is ready, it is often considered easy-to-use. However, as aforementioned in the introduction, there are numerous situations where the web-based or app-based user interface becomes unintuitive. This mostly happens when IoT devices to be controlled are many and hard to be specifically identified in the interface which is virtual. We list a few representative previous studies that tried to make a device specification process more straightforward.

Swindells et al. [27] proposed an IR-based transceiver for a user device (i.e., controller), from which a tiny ping message that wakes up a target device is transmitted and by which the replied target device address information is received back. Using the received information, the transceiver lets the WiFi interface of the user device be automatically connected to its target device, by which the NLOS (Non Line of Sight) communication between them is enabled. Spartacus [26] uses an acoustic technique exploiting the Doppler effect. Spartacus lets a user device emit a continuous audio tone and asks its user to perform a pointing gesture toward a target device. If the target device detects the Doppler shift, then it reports the shift value and its ID back to the user device through a wireless channel. When the maximum shift value is found in the user device, Spartacus assumes that the target device with that value is actually pointed by the user and communicates with that device. In a similar context but to achieve higher precision of pointing, Zhang et al. [28] designed a system called HOBS utilizing an IR emitter attached on a head-mounted device, Google Glass [4]. HOBS lets the IR emitter work as a pointer toward target devices and make all the pointed devices simultaneously to be connected to HOBS via XBee interface [13], especially when the devices are located closely to each other. HOBS then asks its user to browse among the connected devices by giving an input and makes each device reacts upon the selection by blinking an LED attached to the device. Through a visual inspection, HOBS lets the user finally decide who to communicate with. These techniques improve the convenience involved in the device specification and association, their coexistence with the aforementioned CoAP-based control framework is under-explored.

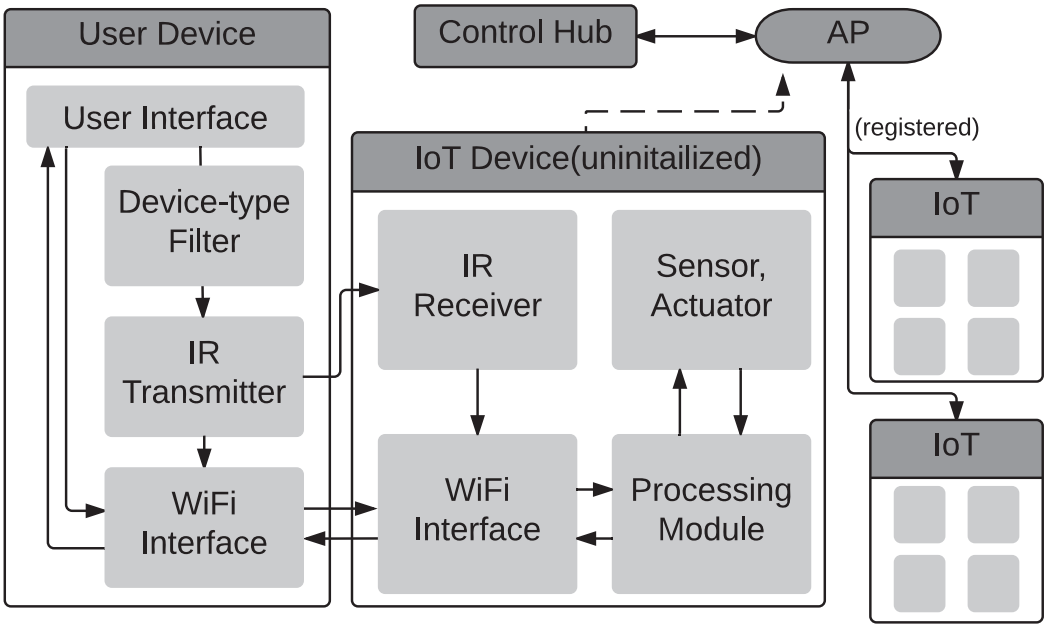


Figure 2: Overview of the architecture of QuickTalk.

3 SYSTEM DESIGN

In this section, we first provide a problem statement that we are trying to solve through this work. Then, with an overview of the software architecture of QuickTalk, we explain the required function blocks of a user device and an IoT device that use QuickTalk. Lastly, we introduce technical challenges involved in enabling QuickTalk in a mobile device.

3.1 Problem Statement

The state of an IoT device can be roughly classified into two categories: 1) *uninitialized* and 2) *registered*. Uninitialized state of an IoT device means that the device is just taken out of its box and it is not currently connected to a control hub (or a control interface provided by the control hub). Every IoT device is given to a user at its uninitialized state and requires the user to go through a certain setup procedure. Once the setup procedure is completed, the state of an IoT device changes to registered and becomes controllable by the associated control interface. When becoming registered, IoT devices in the market are mostly connected to their own control interfaces through WiFi connections. It is important to note that once an IoT device is registered to a control interface, its control permission is given to the person or the group of persons who own the access permission to the control interface. Thus, users who are not reachable to the control interface of an IoT device, are all prevented from controlling the IoT device.

The above-mentioned method of controlling IoT devices though a control interface is reasonable in

general, but gives challenges to an IoT novice when she is visiting a new place and is seeing many IoT devices in the place that are open to any visitor. One way of guiding her to use those IoT devices is to authorize her to access the control interface in which all those IoT devices are pre-registered. However, when the place has a large number of visitors, authorizing individual visitors is not secure given that the control interface is accessible from anywhere through Internet once authorized. Invalidating the authorization when she leaves the place is possible but is of a too much complication. Also, when there are too many of IoT devices in the place, it is difficult for her and other visitors to identify which entity in the control interface corresponds to which actual IoT device.

As a solution to these matters, our goal is to provide an immediate and intuitive method of controlling IoT devices to a user who is in the proximity of those IoT devices. We aim at designing the method to be able to physically pinpoint an IoT device and to work with any IoT device that is either uninitialized or registered.

3.2 QuickTalk Architecture

In order to achieve the objectives, we design the architecture of QuickTalk as it is depicted in Figure 2. To enable QuickTalk between a user device and an IoT device, QuickTalk requires a user device to have an IR transmitter, a WiFi interface, and an IoT control application. Note that because the user device in QuickTalk uses only the IR transmitter to pinpoint an IoT device and does not use IR for communication, it is not necessary to have an IR receiver in the user device. In contrast, an IoT device for QuickTalk needs an IR receiver and a WiFi interface, where the IR receiver is used to detect if the IoT device itself is pointed or not by the IR transmitter and to activate its WiFi when being pointed. More detailed specifications of a user device and an IoT device are provided below.

3.2.1 User Device

For an immediate control of an IoT device from a user device, the user device for QuickTalk utilizes a concept of filter, called device-type filter, and asks a user to provide the type of the IoT device to be controlled when emitting a command to that IoT device. The filter is also helpful to significantly reduce the possibility of experiencing confusion in pointing an IoT device, especially when there are many IoT devices in close proximity. For instance, when trying to control an IoT bulb out of many IoT devices in the same place, a user of QuickTalk may point to the bulb with the filter specified as *BULB* to maximize the pinpointing efficiency. Once a user device succeeds in specifying and activating an IoT device within a very short time, QuickTalk switches to utilize WiFi instead of IR for the reliable (i.e., NLOS) delivery of user commands to the IoT device. We will discuss about the challenges involved in switching to WiFi

in the next subsection.

3.2.2 IoT Device

An IoT device with an IR receiver reacts when an IR signal is detected at its IR receiver. As will be shown later, it is possible to enable QuickTalk without having an IR receiver in an IoT device. However, having an IR receiver which is of low-cost adds two major benefits to an IoT device: 1) intuitive pointing (i.e., specification) and 2) energy efficiency. By the nature of strong directionality of IR signal, when IR transmitter and receiver are properly installed, a very narrow pinpointing ability is achievable. We will discuss about the pinpointing ability of IR signal in the following section. Regarding the energy efficiency, an IR receiver does an important role. Suppose that there is an uninitialized IoT device or an IoT device that is registered but in a power saving mode. In both cases without having an always-on low-power channel that immediately activates the WiFi interface, a user who wants to deliver a command should wait until the WiFi interface becomes active (e.g., until the duty cycle of power saving ends). Thus, installing a small IR receiver in an IoT device that consumes only less than 10mW is a reasonable choice to achieve energy efficiency while keeping the property of immediate control. To improve the energy efficiency, QuickTalk utilizes the device-type filter that arrives at an IoT device inside the pointing IR signal, and selectively activates the IoT device of the matching type (e.g., a *THERMAL Controller* on the same line of sight or IR signal is not activated when a device type is specified as *POWER PLUG*).

3.3 Technical Challenges

There are two major technical challenges that need to be tackled to enable QuickTalk. The first is to reveal the pinpointing ability of IR signal in real IoT environments and to design the frame structure of the IR signal for reliable message delivery. The second is to design and implement a WiFi communication method that can exchange packets either with uninitialized IoT devices or registered IoT devices.

3.3.1 IR Pinpointing

The most important feature of the IR signal to be studied for QuickTalk is its ability of pinpointing an IoT device. Given that most remote controllers using IR work well in an indoor situation, our focus is given more onto the pinpointing ability at an outdoor environment. Also, the reachable distance and the allowed amount of slanted angle for pinpointing are also of our interest. In addition to the study of these properties, engineering decisions related to the channel coding of the information bits in an IR transmission, the types of information bits to be included, and the length of information bits for an IR transmission are all to be discussed and designed in detail. We provide our answers to these matters in

the next section.

3.3.2 Association-Free WiFi Communication

In order to make QuickTalk to be an immediate control method that works universally to any IoT device, a WiFi communication method that works before a user device is associated with an IoT device is essential. Because WiFi association generally requires scanning and authentication procedures, it incurs a non-negligible amount of delay. Thus, it is most desirable if an IoT device can immediately accept a message when it is triggered by an IR signal without going through an association process. We call such a method as *Association-Free* communication. The benefit of association-free communication is not only its latency but also its capability of coexistence with ongoing WiFi communication. How to implement an association-free communication and how to let it give negligible impact to the ongoing communication if any are on our technical challenges, which will be answered in the next section.

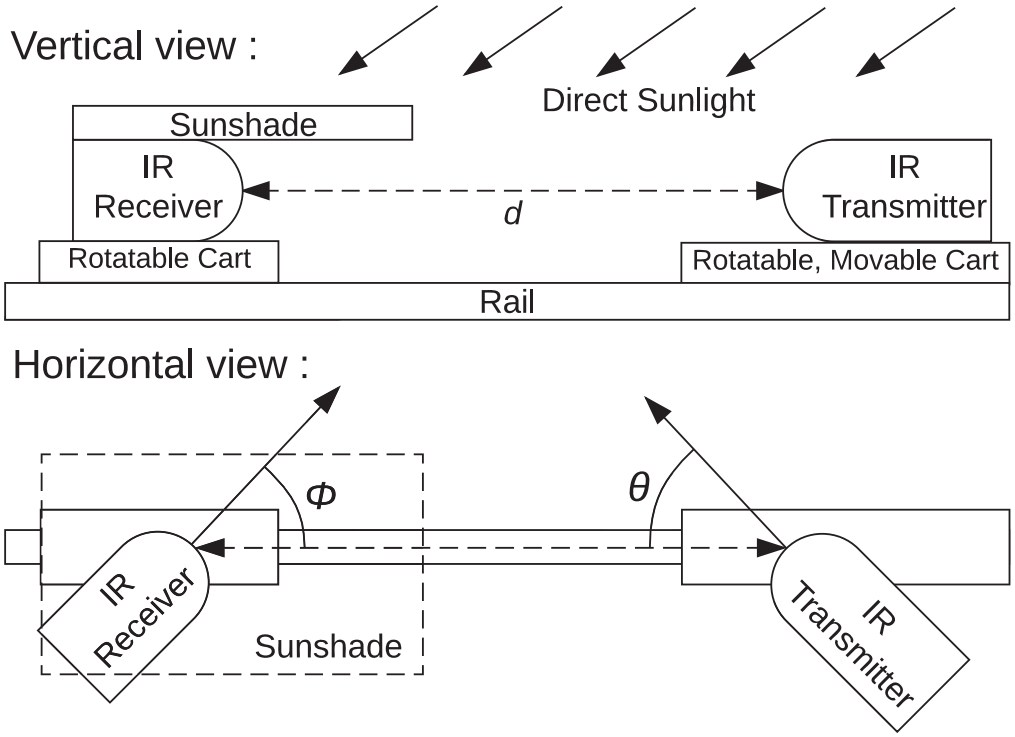


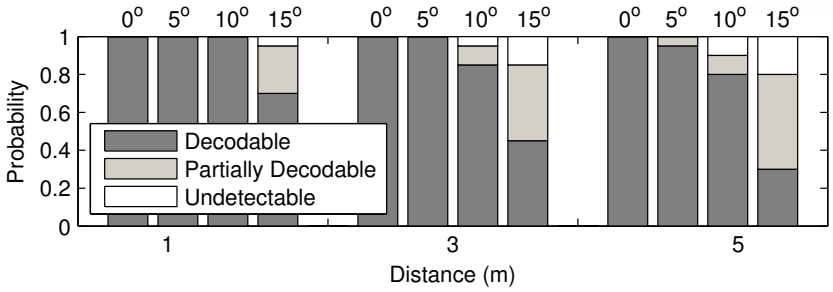
Figure 3: Vertical and horizontal views of the test platform. The distance and angles are controllable as the IR transmitter is on a rotatable and movable cart and the IR receiver is on a rotational cart.

4 PROPOSED METHODS

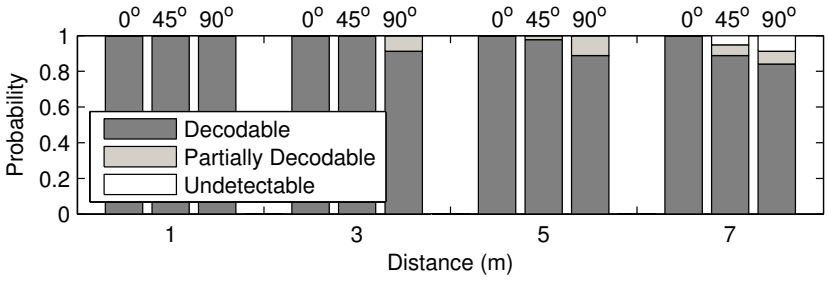
In this section, we propose our systems designs as solutions to the challenges raised in implementing IR pinpointing and association-free communication, and validate our proposed designs using Raspberry Pi version 2 devices with 4.1.19 Linux kernel installed, which emulate both user devices and IoT devices.

4.1 IR Pinpointing

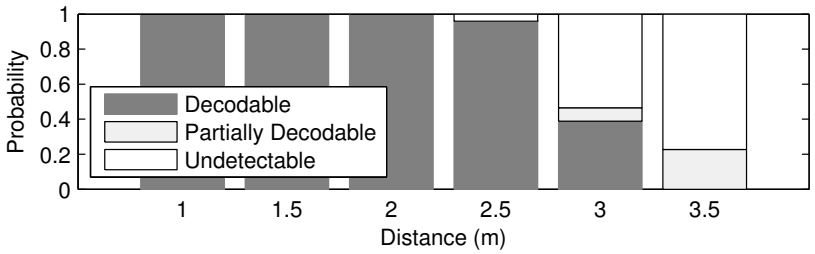
The purpose of IR pinpointing is twofold: 1) to specify an IoT device to control and 2) to trigger the WiFi interface hence enabling association-free communication. In order to achieve both, we design the data frame emitted whenever a user device pinpoints an IoT device. Our design is to emit the ID of the user device in the form of its WiFi MAC address (24bits) and emit device type filters in a hierarchical manner (total 14 bits that are divided into 4, 4, and 6 bits by the levels of categorization) along with parity bits (2bits). In total, each IR signal emission delivers 40 bits of information to an IoT device, which typically takes less than 95 milliseconds. The reason why we have hierarchical device type filters is to maximally narrow down the pointed candidates, hence finally pinpointing a device. For instance, an interactive IoT advertisement display can be hierarchically classified as *DISPLAY:AD-DISPLAY:INTERACTIVE-AD-DISPLAY*. Using the concept of hierarchical filter, a user device may trade off the user convenience (e.g.,



(a) Indoor: Transmission angle (θ) varies while $\phi = 0$.



(b) Indoor: Reception angle (ϕ) varies while $\theta = 0$.



(c) Outdoor: The case of aligned angles, $\theta = \phi = 0$.

Figure 4: The probability distribution of the received IR signal over the cases: decodable, partially decodable, and undetectable, where the IR signal exchange is experimented at indoor with varying (a) transmission angle (θ), (b) reception angle (ϕ), and (c) at outdoor with aligned angles.

user knowledge) of pinpointing and the precision of pinpointing, which is out of the scope of this work and will be of a separate study.

4.1.1 Validation

Using one of de-facto standard of IR communication, called NEC format [2], we validate if our data frame delivery through IR can pinpoint an IoT device. For the validation, we made a test platform as described in Figure 3. As it is shown, it is to easily test various factors such as transmission distance (d), transmission angle (ϕ), and reception angle (θ). By using the rail and the rotational and movable cart, we performed extensive validations on IR transmitter and receiver that are both operated at the modulation rate of 38 KHz. As the NEC format specifies, our IR signal transmission accompanies 13.5ms(9ms of

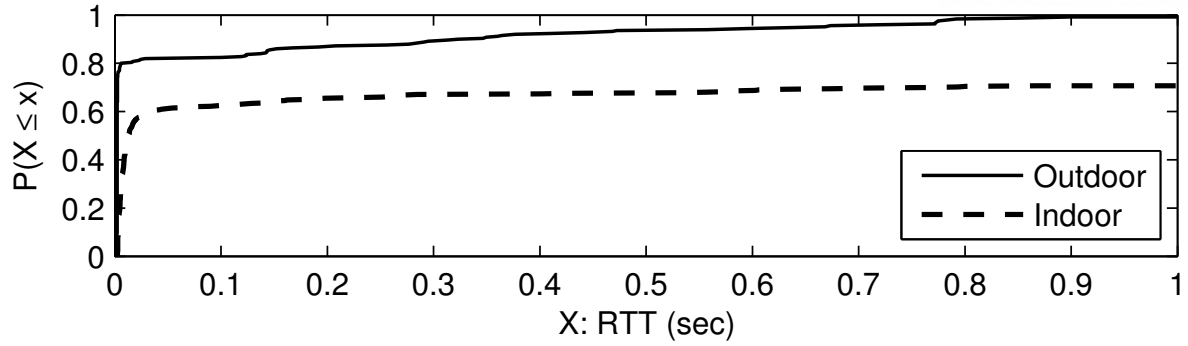
ON period and 4.5ms of OFF period) of lead code emission at the beginning for the purpose of separating each IR signal. After the leader code emission, our data frame of 40 bits is followed. Since we find that the repetition of the entire bits followed by the data frame transmission, which is specified in NEC format does not critically affect the success rate of IR transmissions, we intentionally omitted this repetition part for simplicity. Figure 4 (a), (b), and (c) show that how the data frame sent from a user device is delivered to an IoT device at different settings. We classify the situations by whether the delivered bits are fully decodable, partially decodable, or undetectable. As shown in Figure 4 (a), it is natural to observe that the longer the distance, the narrower the decodable angle is. However, it is important to note that even at the distance of 5 meters, we can manage to narrow down the decodable angle to be less than 10 degree, meaning that IoT devices in 5 meter distance will be activated when they are located at a circle of 88.2 centimeters at that distance. Considering the typical spacing of IoT devices, it is fair to say that it is of a sharp pinpointing. According to a recent study [17], the decodable angle of an IR transmitter can be physically controlled without manufacturing a high-cost transmitter only by adjusting the depth of installing an IR transmitter in its housing. Figure 4 (b) further shows that when it is well pointed by an IR transmitter, the mismatch in the reception angle of an IR receiver does not degrade the decodability. Considering that IoT devices can be installed in various postures, hence having their IR receivers headed toward random directions, our observation in Figure 4 (b) is optimistic to the users who want to remotely control such IoT devices. Finally, Figure 4 (c) reveals that IR signal exchange up to 2.5 meters is even possible at a sunny outdoor environment as long as an IR receiver is shaded from the direct sunlight. Overall, we confirm that our IR system design is practically viable in pinpointing an IoT device.

4.2 Association-Free WiFi Communication

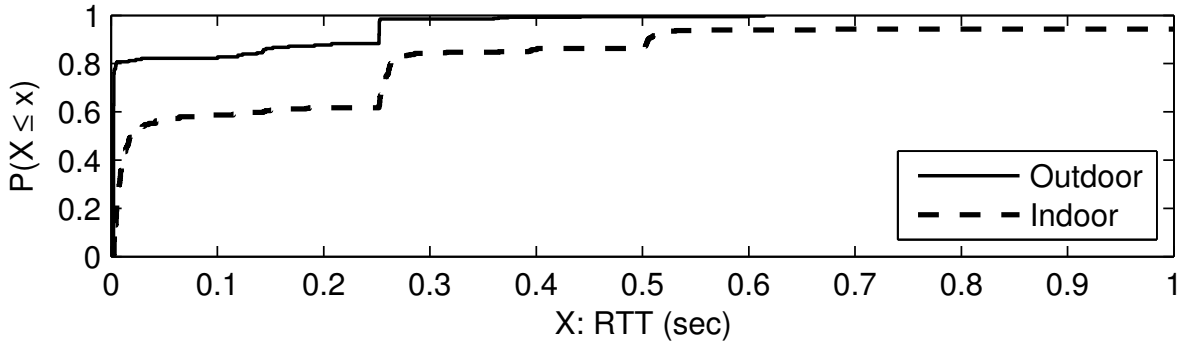
To our knowledge, there can be two different ways of implementing a WiFi communication method for QuickTalk, which can coexist with established WiFi sessions if any.

The first is to use a *fake PS poll*, where PS poll stands for the power save poll defined in the 802.11 standard [6]. The main idea behind this method is to let an IoT device previously registered to a control interface send a fake PS poll to the access point that the IoT device is being associated upon reception of IR signal. When a fake PS poll is received at the access point, it is known by a work [18] that the access point suspends all the ongoing sessions and queue the undelivered packets in the access point. Thus, the IoT device is able to secure a certain amount of period that can be used to communicate with a user device who triggered the IoT device by an IR signal. This is a working method but is of a hack, which is not recommended.

The second is to use the intrinsic packet broadcast ability and the packet monitoring (i.e., capturing) ability of WiFi. We name such a method as *association-free communication*. To enable association-free



(a) Without retransmission



(b) With retransmission

Figure 5: The CDFs of RTT from the association-free communication exploiting packet broadcasts. (a) When there is no application-level retransmission, about 93.6% and 67.6% of packets are replied within 0.5 seconds at an outdoor and an indoor environment, respectively. (b) The percentages increase to 99.5% and 86.5% when adding the application-level retransmission that retries after 0.25 seconds.

communication, upon reception of an IR signal, we let the IoT device broadcast the MAC address of the user device which is received through the IR signal and let the user device go to the packet monitor mode and switch WiFi channels to detect in which channel the broadcast MAC address is received back. Once the channel is identified, we let the user device also send out its data packets and commands through the same broadcast method and let the IoT device do the same for data exchange. Because packet broadcasts can coexist with any ongoing WiFi sessions, the coexistence of immediate communication in proximity and communication through a control interface is guaranteed. We adopt this association-free communication as our default communication method for QuickTalk.

4.2.1 Validation

To validate the proposed association-free communication, we perform the following experiment either at an indoor environment and at an outdoor environment. The experiment is to measure the RTT (Round-Trip Time) between the broadcast of a random payload of 24 bits at every 3 seconds from a user device to an

IoT device and its reply of the same payload to the user device from the IoT device. For this experiment, we first do not use any application-level packet retransmission scheme in order to identify the pure performance of the proposed method [1] and keep the signal strength between the user device and the IoT device between -30 and -60 dBm.

Figure 5 (a) shows the CDFs (Cumulative Density Functions) of RTT measured either at an indoor and an outdoor environment when there is no application-level packet retransmission. As shown in the figure, at an outdoor environment where there is almost no interfering WiFi signal, about 80% of broadcast packets are successfully replied within in 0.01 seconds. At an indoor environment where we were able to scan about 30 interfering WiFi access points, within 0.01 seconds of RTT, about 60% of broadcast packets are successfully replied. We further validate the performance of association-free communication when an application-level packet retransmission is implemented. Figure 5 (b) shows the CDFs of RTT measured either at an indoor and an outdoor environment when we set the retransmission happens at every 0.25 seconds when the packet reception is not successful. As shown in the figure, we can observe substantial improvement in the reply rate as in both indoor and outdoor environments more than 85 % of broadcast packets are replied within 0.5 seconds, confirming that the association-free communication is practically viable.

[1] Although there is no application-level retransmission, link-level retransmissions from 802.11 standard may work.

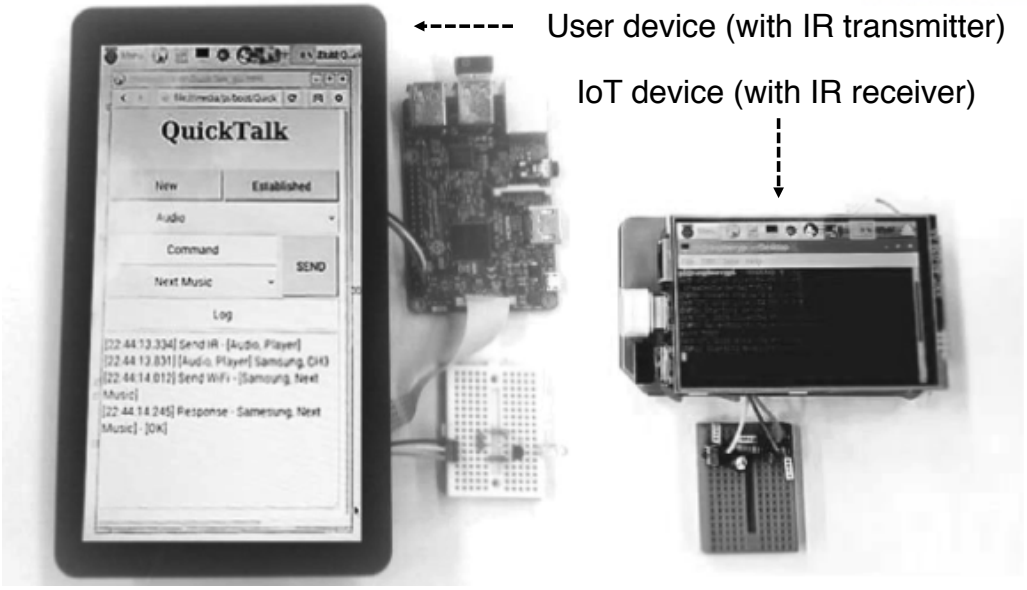


Figure 6: QuickTalk implementation for a user device (left) and for an IoT device (right). The screen attached to the user device shows our user interface.

5 IMPLEMENTATION

In this section, we present implementation detail of a user device and an IoT device that use QuickTalk. As it is aforementioned, we exploit the Raspberry Pi 2 platform with IR circuits connected through GPIO (General Purpose Input/Output) in order to prototype both devices as shown in Figure 6.

5.1 User Device Implementation

Our implementation of user device consists of two parts: 1) user interface and 2) IR and WiFi services. The user interface is designed to get commands from a user either by clicking buttons, by typing commands, or by voice commanding. Our graphical user interface (GUI) is currently implemented by HTML and C++ using CGI (Common Gateway Interface) [23] and the voice commanding function uses Google speech recognition APIs [3]. For an immediate commanding to an IoT device, our user interface asks to provide the device type information along with a command. When the type information and the command are given, IR service first sends out the device type information and the MAC address of the user device as described in Section 4, then WiFi service captures (i.e., monitors) the MAC address broadcasted by the triggered IT device and delivers the command to that IoT device. Including this initial command delivery, all the following data exchanges through packet broadcasts use CoAP format provided Californium (Cf) JAVA library [20].

Figure 6 shows how the user interface implemented on a Raspberry Pi 2 device is presented to a user. IR and WiFi services therein are both implemented by C++ and use LIRC (Linux Infrared Remote

Algorithm 1 User device algorithm

```
1: procedure IR SERVICE
2:     (category_in, command_in) = userinput()
3:     sendEncodedIRMessage(category_in)
1: procedure WIFI SERVICE
2:     if !channelDetected() then
3:         setRandomChannel()
4:         for i = 1 to CHANNELS do
5:             setNextChannel()
6:             if (ch_IoT, MAC_user)= receiveResponse() then
7:                 break
8:     broadcastMessage(command_in, ch_IoT)
9:     startPacketMonitor()
10:    while TRUE do
11:        if response_IoT = receiveResponse() then
12:            displayResult(response_IoT)
13:            break
14:        if needRETRANSMISSION() then
15:            broadcastMessage(command_in, ch_IoT)
```

Control) API for the operations of IR functions and socket API and MediaTek driver API for the packet broadcast and broadcast packet capture. The driver API is currently limited to the chipsets of MediaTek which includes MediaTek MT7601U (802.11 b/g/n) chipset that we connected to Raspberry Pi 2 devices through USB, but it is possible to extend the API for other WiFi chipsets. Upon capturing a broadcast packet, our WiFi service utilizes the packet capture library, *libpcap*, to extract the contents from the broadcast packets and to detect the identity (i.e., WiFi MAC address) of the triggered IoT device.

Because the IR module in the user device for QuickTalk is intentionally designed not to receive any information through IR communication, how to find the channel where the triggered IoT device broadcasts packets is of a challenge. To tackle this problem, our WiFi service is designed to randomly choose a channel and sweeps the channels one by one for two times. Having two runs of sweeping is to reliable detect the channel where the IoT device is in. We describe the overall procedures that a user device goes through for commanding an IoT device as a pseudo code in Algorithm 1.

Algorithm 2 IoT device algorithm

```
1: procedure IR SERVICE
2:     MAC_user = parseIRMessage()
3:     if !checkParity() or !checkCategory() then
4:         endProcedure()
5:     else
6:         startWiFiservice()
1: procedure WIFI SERVICE
2:     while SWEEPING_TIME_OUT do
3:         broadcastMessage(MAC_user, MAC_IoT)
4:         startPacketMonitor()
5:         wait(BROADCAST_INTERVAL)
6:         if command_user = commandReceived() then
7:             response_out = processCommand(command_user)
8:             broadcastMessage(response_out)
9:             break
```

5.2 IoT Device Implementation

Our implementation of an IoT device consists of two parts: 1) data processing service and 2) IR and WiFi services. The data processing service serves as the core of each IoT device, where the user command is processed and responded. The data processing service is also implemented by Californium library because it parses the CoAP queries and packages data in the CoAP format. Our implementation of the data processing service is general in that it can adapt to any form of IoT devices by a simple modification. Our current prototyping of IoT devices includes IoT bulb, IoT advertisement display, and IoT environmental sensor but not limited to those. The data processing service interacts with IR and WiFi services through the loopback interface.

The IR service in an IoT device is to receive an IR signal which include the MAC address of a user device and the hierarchically designed device type filters and triggers the WiFi service to broadcast the MAC address when the device type filters match with the properties of the IoT device itself. For the reception of broadcast packets, the WiFi service in the IoT device also uses MediaTek driver API to active the monitor mode (i.e., packet capture mode). We describe the overall procedures that an IoT device goes through as a pseudo code in Algorithm 2.

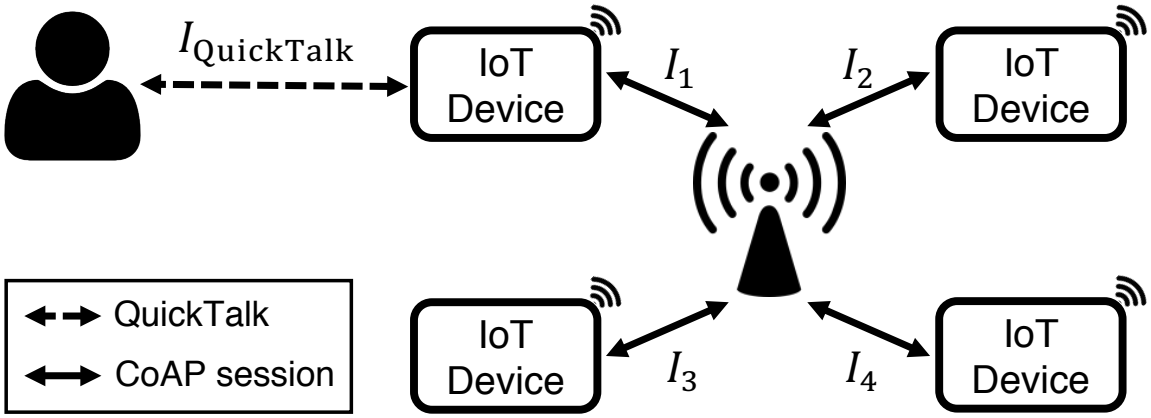


Figure 7: The topology used for the evaluations of QuickTalk. I_i denotes the packet arrival rate (packets per second) of i-th ongoing CoAP communication session whereas $I_{\text{QuickTalk}}$ stands for the packet arrival rate of QuickTalk.

6 EVALUATION

We evaluate the performance of QuickTalk in a challenging situation where the candidate IoT devices to be controlled are previously registered to a control hub and are communicating with the hub through an WiFi access point as depicted in Figure 7. We assume that there are 4 registered IoT devices and a user device tries QuickTalk to one of those IoT devices. In such a situation, we first test if QuickTalk indeed enables an immediate commanding to an IoT device by measuring the end-to-end delay of QuickTalk, and then we further test how much performance degradation of the ongoing sessions between the IoT devices and the control hub experience when a user device communicates with one of the IoT devices through QuickTalk.

6.1 End-to-end delay of QuickTalk

The end-to-end delay of QuickTalk from its IR signal transmission to the WiFi packet reception of the acknowledgement for an IoT command is mainly composed of two delay components: T_{search} and $T_{\text{broadcast}}$ as depicted in Figure 8. Here, T_{search} denotes the time duration of scanning channels to detect in which channel the IoT device triggered by an IR signal makes the broadcast. Since we set the channel switching duration as 40 ms in our experiment, the worst case of T_{search} becomes 2 seconds given that our design takes two rounds of channel sweeping. The average delay for T_{search} simply becomes the half of the worst case value. When the channel of interest is detected, the remaining delay is determined by $T_{\text{broadcast}}$ where $T_{\text{broadcast}}$ denotes the time duration between sending a command and receiving its acknowledgement through the broadcast channel of the WiFi interface. There are other delays in QuickTalk such as

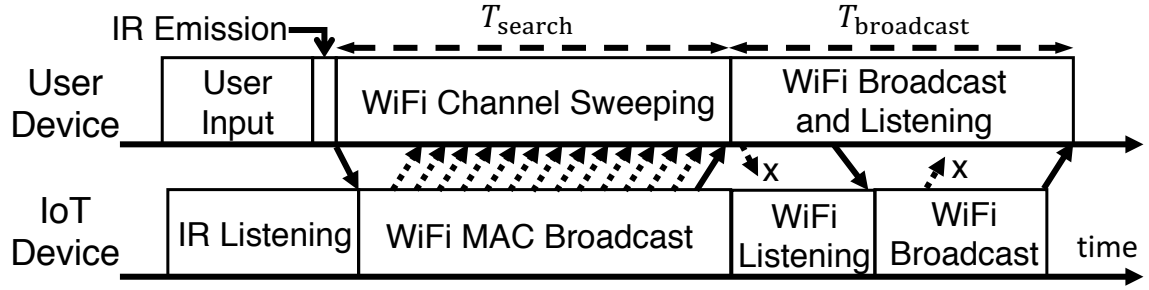
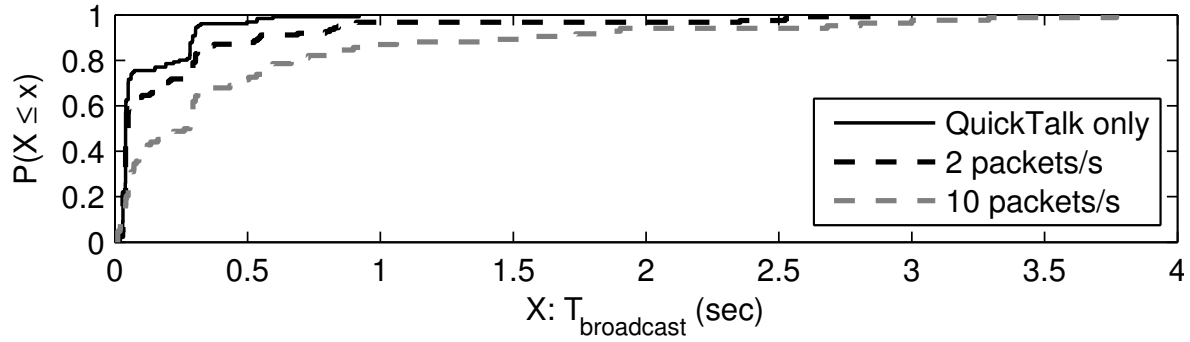
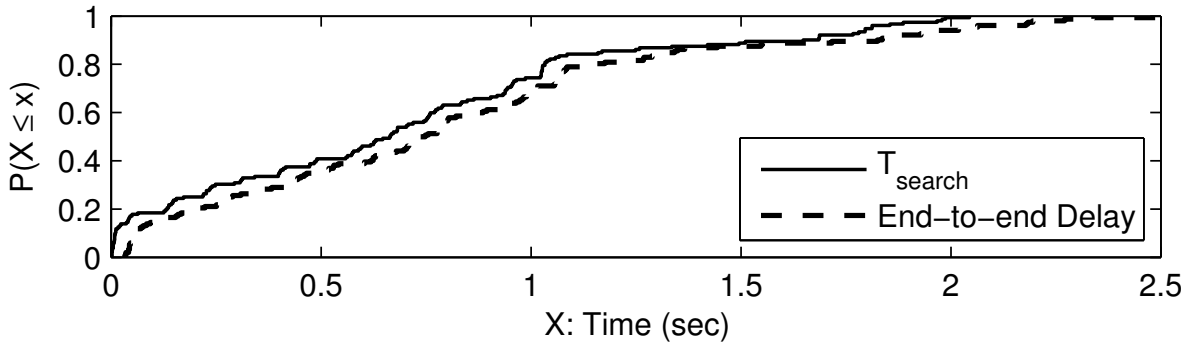


Figure 8: We find that the end-to-end delay of QuickTalk is mainly affected by two major components: T_{search} and $T_{\text{broadcast}}$.



(a) CDFs of $T_{\text{broadcast}}$ with competing CoAP sessions

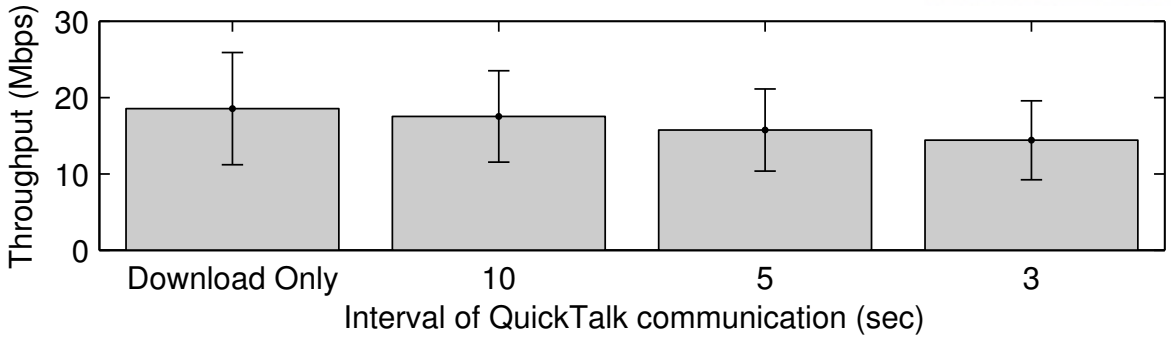


(b) CDFs of T_{search} and the end-to-end delay

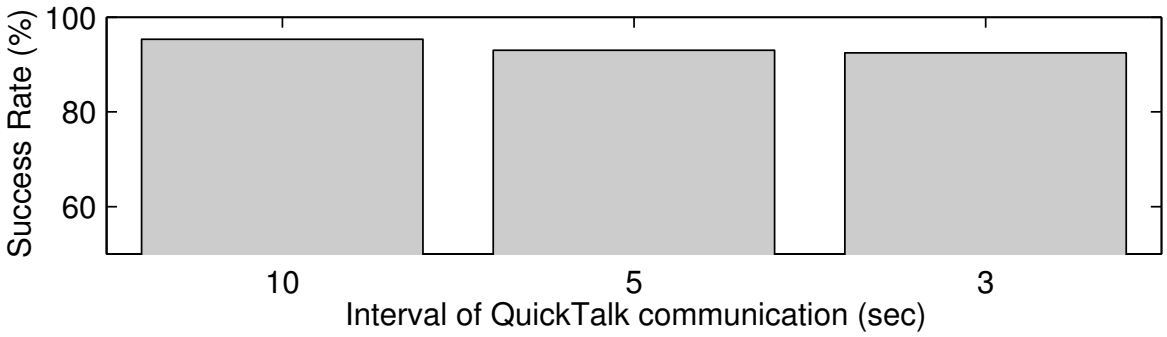
Figure 9: CDFs of (a) $T_{\text{broadcast}}$ when there are 4 competing CoAP sessions that have 2 or 10 packets per second for each session and (b) T_{search} and the end-to-end delay. The end-to-end delay of QuickTalk has its median at 0.74 seconds.

processing time for an IoT command, packet extraction time from a broadcast channel, and context switching delay from the IR service to the WiFi service, but we find that all such delays are in the scale of a few milliseconds in the processor of Raspberry Pi 2 (ARM Cortex A7, Quad-core, 900 MHz). Thus, we mainly focus on T_{search} and $T_{\text{broadcast}}$.

Figure 9 (a) shows the CDF of $T_{\text{broadcast}}$ when there is no ongoing CoAP session or when there are 4



(a) Average download throughput



(b) Success rate of QuickTalk

Figure 10: (a) The throughput of an download session at an IoT device and (b) the success rate of QuickTalk communication with that IoT device when the download session coexists with QuickTalk of various communication intervals.

ongoing sessions as shown in Figure 7. For each ongoing session, we vary the packet arrival rate (i.e., I_i) by 2 and 10 packets per second. As Figure 9 (a) confirms, QuickTalk mostly experiences less than 0.5 seconds for $T_{\text{broadcast}}$ when there is no ongoing CoAP session and experiences less than 1 second for $T_{\text{broadcast}}$ at 80% of the cases, when there are 4 busy CoAP sessions that exchange 10 packets per second each.

Figure 9 (b) shows the CDF of T_{search} where is no ongoing CoAP session. Because we observe that having a number of ongoing CoAP sessions does not affect T_{search}, we only present the result with no ongoing CoAP session. As shown in Figure 9 (b), T_{search} is relatively widely distributed from 0.04 seconds to 2 seconds since our channel sweeping algorithm naively starts from a randomly chosen channel. Note that we can linearly speed up T_{search} by reducing the channel switching delay, but for this, the hardware support is essential as the switching delay is currently bounded by the chipset delay. Also note that T_{search} can be completely eliminated when a user device for QuickTalk is redesigned to receive IR signal from an IoT device regarding its current WiFi channel, but we consider that this is not user-friendly since this compels the user to keep its posture until the IR signal is successfully returned.

The end-to-end delay of QuickTalk is also presented in Figure 9 (b) for the case where there is no ongoing CoAP session. As aforementioned, we find that the end-to-end delay is not much different from $T_{\text{search}} + T_{\text{broadcast}}$ and is upper limited by 2.5 seconds while its median is only about 0.74 seconds. Note that once the WiFi channel is detected, the end-to-end delay of QuickTalk approaches to $T_{\text{broadcast}}$, which is roughly upper bounded by 1 second.

6.2 Coexistence with Ongoing Sessions

The coexistence of QuickTalk with ongoing communication sessions at an IoT device is an important matter given that there can be many IoT devices in practice, which are previously registered to control hubs but need to be immediately controlled by a user in proximity. To test the coexistence, we let one IoT device shown in Figure 7 perform TCP-based file download and evaluate how much throughput degradation is observed when QuickTalk starts communicating with that IoT device for various commanding intervals, 10, 5, and 3 seconds. Figure 10 (a) shows the download throughput with 95% confidence interval, which is measured at the IoT device without and with QuickTalk. Figure 10 (b) further shows the success rate of QuickTalk communication with an ongoing download session for various QuickTalk communication intervals. As the graphs show, the degradation of the throughput is limited to about 20% when QuickTalk commands at every 3 seconds, compared to the download only case. Also, the success rate of QuickTalk communication stays over 92% while the interval varies from 10 seconds to 3 seconds. This experiment reveals that QuickTalk can reliably coexist with ongoing communication sessions in IoT devices.

7 CONCLUDING REMARKS

In this work, we proposed QuickTalk, an association-free communication method for IoT devices in proximity that is designed to enable intuitive, immediate and pinpointed communications with IoT devices around an IoT user. Our implementation of QuickTalk using Raspberry Pi 2 devices confirms that QuickTalk works reliably in realistic environments and further shows that its end-to-end delay for delivering a command is reasonably low with the worst case bound of 2.5 seconds. We believe that QuickTalk that can be activated in every IoT device only by adding an IR receiver of a few cents can give a whole new user experience for IoT devices especially to non-tech savvy users.

References

[1] ARENA Solutions. Connecting devices to the Internet of Things with Wi-Fi, 2015. http://embedded-computing.com/white-papers/white-beyond-bom-101/.

[2] Data Formats for IR Remote Control, 2013. **http://www.vishay.com/docs/80071/dataform.pdf**.

[3] Google Cloud Speech API. **https://cloud.google.com/speech/**.

[4] Google Glass. **https://developers.google.com/glass/**.

[5] How the AWS IoT Platform Works. **https://aws.amazon.com/iot/how-it-works/?nc1=f_ls**.

[6] IEEE std 802.11e-2005. *IEEE Standard for Information technology–Local and metropolitan area networks–Specific requirements–Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications*.

[7] LG U+ launches "U+ Gaslock", the Home IoT Service enabling Gas Valve Remote-Control, 2014. **http://www.netmanias.com/en/post/korea_ict_news/7004/iot-lg-u/**.

[8] Nest. **https://nest.com/thermostat/meet-nest-thermostat/**.

[9] Neurio. **http://neur.io/products/**.

[10] Philips hue. **http://www.developers.meethue.com/documentation/how-hue-works**.

[11] QRCode. **http://www.qrcode.com/en/about/**.

[12] Use the Home app on your iPhone, iPad, and iPod touch. **https://support.apple.com/en-us/HT204893**.

[13] XBee: Connect Devices To The Cloud - Digi International. **https://www.digi.com/lp/xbee**.

[14] ATZORI, L., IERA, A., AND MORABITO, G. The internet of things: A survey. *Elsevier Computer networks 54*, 15 (2010), 2787–2805.

[15] BETTS, D. Microsoft Azure IoT services: reference architecture. **https://azure.microsoft.com/en-us/documentation/articles/iot-suite-what-is-azure-iot**.

[16] BORMANN, C., STUREK, D., AND SHELBY, Z. 6LowApp: Problem Statement for 6LoWPAN and LLN Application Protocols, 2009. **https://tools.ietf.org/html/draft-bormann-6lowpan-6lowapp-problem-01**.

[17] CHEN, Y.-H., ZHANG, B., TUNA, C., LI, Y., LEE, E. A., AND HARTMANN, B. A context menu for the real world: Controlling physical appliances through head-worn infrared targeting. Tech. rep., 2013.

[18] GU, W., YANG, Z., QUE, C., XUAN, D., AND JIA, W. On security vulnerabilities of null data frames in ieee 802.11 based wlans. In *IEEE ICDCS* (2008).

[19] GUINARD, D., AND TRIFA, V. Towards the web of things: Web mashups for embedded devices. In *Workshop on Mashups, Enterprise Mashups and Lightweight Composition on the Web* (2009).

[20] KOVATSCH, M., LANTER, M., AND SHELBY, Z. Californium: Scalable cloud services for the internet of things with coap. *IEEE International Conference on the Internet of Things* (2014).

[21] KOVATSCH, M., MAYER, S., AND OSTERMAIER, B. Moving application logic from the firmware to the cloud: Towards the thin server architecture for the internet of things. In *IEEE International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing* (2012).

[22] LEVÄ, T., MAZHELIS, O., AND SUOMI, H. Comparing the cost-efficiency of CoAP and HTTP in web of things applications. *Elsevier Decision Support Systems 63* (2014), 23–38.

[23] ROBINSON, D. The common gateway interface (CGI) version 1.1, 2004. **https://tools.ietf.org/html/rfc3875**.

[24] SHELBY, Z., HARTKE, K., AND BORMANN, C. The constrained application protocol (CoAP). Tech. rep., 2014.

[25] SHELBY, Z., STUBER, M. G., STUREK, D., FRANK, B., AND KELSEY, R. CoAP feature analysis, 2009. **https://tools.ietf.org/html/draft-shelby-6lowapp-coap-00**.

[26] SUN, Z., PUROHIT, A., BOSE, R., AND ZHANG, P. Spartacus: spatially-aware interaction for mobile devices through energy-efficient audio sensing. In *ACM MobiSys* (2013).

[27] SWINDELLS, C., INKPEN, K. M., DILL, J. C., AND TORY, M. That one there! pointing to establish device identity. In *ACM UIST* (2002).

[28] ZHANG, B., CHEN, Y.-H., TUNA, C., DAVE, A., LI, Y., LEE, E., AND HARTMANN, B. HOBS: head orientation-based selection in physical spaces. In *ACM symposium on Spatial user interaction* (2014).